Important Instructions to examiners:

1) The answers should be examined by key words and not as word-to-word as given in the model answer scheme.

2) The model answer and the answer written by candidate may vary but the examiner may try to assess the understanding level of the candidate.

3) The language errors such as grammatical, spelling errors should not be given more Importance (Not applicable for subject English and Communication Skills).

4) While assessing figures, examiner may give credit for principal components indicated in the figure. The figures drawn by candidate and model answer may vary. The examiner may give credit for any equivalent figure drawn.

5) Credits may be given step wise for numerical problems. In some cases, the assumed constant values may vary and there may be some difference in the candidate's answers and model answer.

6) In case of some questions credit may be given by judgment on part of examiner of relevant answer based on candidate's understanding.

7) For programming language papers, credit may be given to any other program based on equivalent concept.

Marks

1. Attempt any TEN of the following : 20

(a) Write any two characteristics of procedure oriented programming.

(Any 2 characteristics, Each characteristic – 1 Mark)

Ans:

1. Emphasis is on data rather than procedure.
2. Programs are divided into objects.
3. Data structures are designed such that they characterize the objects.
4. Functions that operate on data of an object are tied together in the data structure.
5. Data is hidden and cannot be accessed by external functions.
6. Objects communicate with each other through functions.
7. New data and functions can be easily added whenever necessary.
8. Follows bottom-up approach in program design.

(b) Define class with it's syntax.

(Class definition – 1 Mark, Syntax – 1 Mark)

Ans: **Definition:** a class is a way of binding data and its associated functions together.

Syntax:

```
        classclass_name
{
private:
        Variable declarations;
        Function declarations;
public:
Variable declarations;
        Function declarations;
};
```

(c) Write any two rules to define friend function.

(Any two rules, Each rule – 1 Mark)

Ans: **Rules to define friend function:**

1. It is not in the scope of the class to which it has been declared as friend.
2. As it is not in the scope of the class, it cannot be called using the object of that class.
3. It can be invoked like a normal function without the help of any object.
4. It cannot access the member names directly and has to use an object name and dot membership operator with each member name.
5. It can be declared either in the public or the private part of a class without affecting its meaning.
6. It has the objects as arguments.

(d) What is copy constructor?

(Define – 1 Mark, for any relevant point - 1 Mark)

Ans: Copy constructor is used to declare and initialize an object from another object.

Example: student s1 (10); - object s1 is created and initialized student s2 (s1);- copy constructor is called and object s2 is initialized with values of object s1.

(e) State different visibility modes used in inheritance.

(Three modes – 2 Marks)

Ans: There three visibility modes used in inheritance:

1. Private
2. Protected
3. Public

(f) What is pure virtual function?

(Define – 1 Mark, For any relevant point - 1 Mark)

Ans: A pure virtual function is a function declared in a base class that has no definition relative to the base class.

Example: virtual void display () =0;

(g) Define polymorphism. List type of polymorphism.

(Definition – 1 Mark, Types – 1 Mark)

Ans: Polymorphism means one name, multiple forms. Same named functions behave differently depending on type of data used in operation.

Types:-

1. Compile time polymorphism
2. Run time polymorphism

(h) What is the significance of scope resolution operator?

(Use of scope resolution operator – 2 Marks)

Ans: Scope resolution operator allows access to the global version of a variable, and also allows facility to define member functions outside of a class.

It is represented with double colon (::).

(i) Define pointer variable. Give its syntax.

(Definition – 1 Mark, Syntax – 1 Mark)

Ans: A pointer is variable that stores memory address of another variable of similar data type.

Syntax: - data_type * pointer_variable_name;

(j) **Define a structure with it's syntax.**

(Definition – 1 Mark, Syntax – 1 Mark)

Ans: Structure is a collection of different data types written under a common name. It is a user defined data type.

Syntax:-

```
structstructure_name
{
        Data_type variable 1;
        Data_type variable 2;
                •
                •
                •
        Data_type variable n;
};
```

(k) **List any four object oriented languages.**

(Each language – ½ Mark)

Ans: object oriented languages:

1. Simula
2. Smalltalk
3. C++
4. Ada
5. Java
6. C#

(l) **Write any two characteristics of static member function.**

(Any 2 characteristics, Each characteristic – 1 Mark)

Ans: Characteristics of static member function:

1. A static member function can have access to only other static members declared in the same class.
2. A static member function can be called using the class name:

   Class_name::function_name;

(m)What is an abstract base class?

(Definition –2 Marks)

Ans: An abstract class is a class that is not used to create objects. It is designed only to act as a base class.

(n) How do we invoke a constructor?

(Description with example – 2 Marks)

Ans: Constructor is invoked whenever an object of its associated class is created.

Example:

```
classabc
{
public:
        abc()
        {
        }
};
void main()
{
abc a;  //invokes constructor
}
```

2. Attempt any FOUR of the following: 16

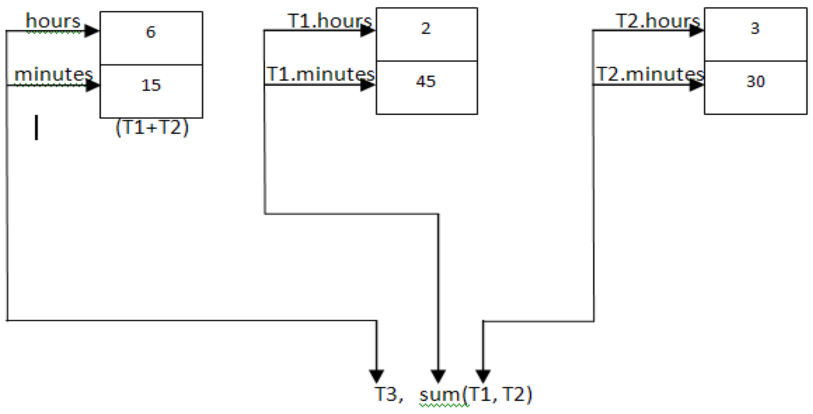
(a) Explain how memory is allocated to an object of a class with diagram.

(Description – 2 Marks, Diagram – 2 Marks)

Ans: Description:

Memory space for objects is allocated when they are declared. The member functions are created and placed in the memory space only once when they are defined as a part of class specification. Since all the objects belonging to that class use the same member functions, no separate space is allocated for member functions when the objects are created. Only space for member variables is allocated separately for each object.

Diagram:



(b) Write any four rules to define constructor in a class.

(Each rule – 1 Mark)

Ans: Rules to define constructor:

1. Constructors should be declared in the public section.
2. They are invoked automatically when the objects are created.
3. They do not have return type, not even void and therefore they cannot return values.
4. They can accept arguments.
5. They cannot be inherited, though a derived class can call the base class constructor.
6. They cannot be virtual.
7. One cannot refer to their addresses.
8. An object with a constructor cannot be used as a member of a union.
9. They make implicit calls to the operators new and delete when memory allocation is required.

(c) Write a program to find reverse of a string using pointer to string.

(***Correct logic – 2 Marks, Syntax – 2 Marks***)

Ans:
```
#include<iostream.h>
#include<conio.h>
#include<string.h>
void main()
{
char str1[10],*ptr;
int l=0;
cout<<"enter string:";
cin>>str1;
ptr=&str1[0];
while(*ptr!='\0')
{
l++;
ptr++;
}
while(l>0)
{
ptr--;
cout<<*ptr;
l--;
}
getch();
}
```

(d) Difference between OOP and POP.

(Any four points are expected, Each point –1 Mark)

Ans:

Procedure oriented programming	Object oriented programming
1.Focus is on procedure.	1. Focus is on data.
2.Large programs are divided into multiple functions.	2.Programs are divided into multiple objects.
3.Data move openly around the system from function to function.	3. Data is hidden and cannot be accessed by external functions.
4.Functions transform data from one form to another by calling each other.	4. Objects communicate with each other through function.
5.Employs top-down approach in program design.	5. Employs bottom-up approach in program design
6.Procedure oriented approach is used in C language.	6. Object oriented approach is used in C++ language.

(e) Write a program to overload binary ++ operator.

(Any relevant program shall be considered correct logic – 2 Marks, syntax - 2 Marks)

Ans:

```
#include <iostream.h>
class temp
{
private:
int count;
public:
temp():count(5){  }
void operator ++() {
count=count+1;
}
void Display() { cout<<"Count: "<<count; }
};
int main()
{
temp t;
   ++t;        /* operator function void operator ++() is called */
t.Display();
return 0;
}
```

OR

```
#include <iostream.h>
class complex
{
        float x;
float y;
public:
        complex() { }
complex( float real, float imag)
{ x= real; y=imag; }
complex operator + (complex);
void display (void);
};
complexcomplex: : operator+ (complex c)
{
complex temp;
temp.x=x+c.x;
temp.y=y+c.y;
return(temp);
}
void complex : : dispalay (void)
{
```

```
cout<< x<<" +j"<< y<<"\n";
}
int main(0
{
complex c1,c2,c3;
c1=complex(2.5, 3.5);
c2=complex(1.6, 2.7);
c3=c1+c2;
cout<<"c1="; c1.dispaly();
cout<<"c2="; c2.dispaly();
cout<<"c3="; c3.dispaly();
return 0;
}
```

(f) Write a program that illustrates multiple inheritance.

(Declaration of more than 1 class – 2 Marks, declaration of child class – 1 Mark, calling functions with child class object – 1 Mark)

Ans:

```
#include<iostream.h>
#include<conio.h>
class A
{
protected:
int x;
public:
voidgetA()
{
cout<<"enter x";
cin>>x;
}
};
class B
{
protected:
int y;
public:
voidgetB()
{
```

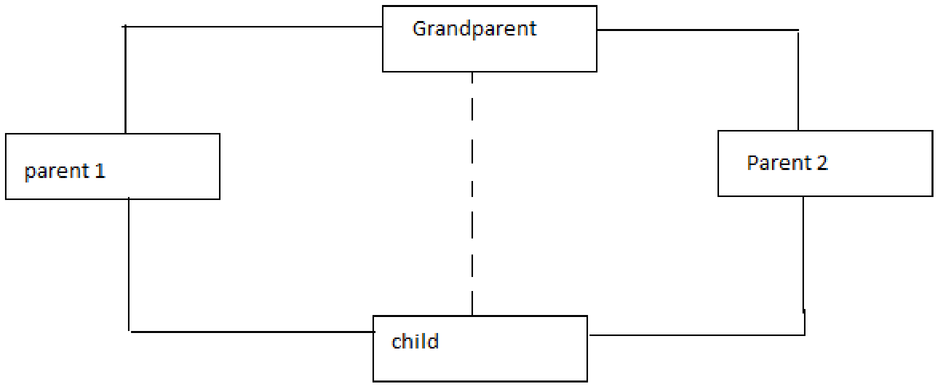
```
cout<<"enter y";
cin>>y;
}
};
class C:public A,public B
{
public:
void display()
{
cout<<"x="<<x;
cout<<"y="<<y;
}
};
void main()
{
C z;
clrscr();
z.getA();
z.getB();
z.display();
getch();
}
```

3. Attempt any FOUR of the following: 16

(a) Explain virtual base class in inheritance with suitable diagram.

(Description – 2 Marks, Diagram – 2 Marks)

Ans: Consider a situation where all three kinds of inheritance, namely, multilevel, multiple, hierarchical inheritance, are involved. This illustrated in fig a. the child has two direct base classes „parent1 & parent2 which themselves have a common base class “grandparent”. The child inherits the traits of “grandparent” via two separate path .it can also inherit directly as shown by broken line. The “grandparent” sometimes referred to as indirect base class.



inheritance by the "child as shown in fig a might pose some problems. All the public & protected members of "grandparent are inherited into "child" twice, first via "parent1& again via "parent 2.This means, "child would have duplicate sets of the members inherited from "grandparent. This introduces ambiguity & should be avoided. The duplication of inherited members due to these multiple paths can be avoided by making the common base class as virtual base class while declaring the direct or intermediate base classes as shown below.

```
 Class A //grandparent
{

//body of class

};

 Class B1: virtual public A //parent1

 {

//body of class

};

Class B2: public virtual A //parent2

{
```

```
//body of class

};

Class C:public B1,public B2 //child

{

//only one copy of A will be inherited //body of class

};
```

(b) Differentiate between compile time polymorphism and runtime polymorphism.
(Any four points each – 1 Mark)

Ans:

Compiletime Polymorphism	Runtime Polymorphism
It simply means that an object is bound to its function call at compile time.	It simply means that selection of appropriate function is done at run time
Functions to be called are know well before	Function to be called is unknown until appropriate selection is made.
This does not require use of pointers to objects	This requires use of pointers to object
Function calls are faster	Function calls execution are slower
Also called as early binding	Also called as late binding
e.g. overloaded function call	e.g. virtual function
It also referred as Static Binding	It also referred as Dynamic Binding

(c) Write program to define a class student having data members name and roll no. Accept and display data for one object.
(Class definition - 2 Marks, creating object t - 1 Mark, Function call - 1 Mark)

Ans:
```
#include<iostream.h>

 #include<conio.h>
```

```
class student
{
private:
introll_no;
char name[20];
public :
void get()
{
cout<<"\nenterrollno& name";
cin>>roll_no>>name;
 }

void put()
{
cout<<"\nroll_no\t" <<roll_no;
cout<<"\nname\t"<<name;
}
};

void main()
 {
student s1;

s1.get();
s1.put();
getch();
}
```

(d) Write a program to calculate area of circle and area of rectangle using function overloading.
(Function for Calculating area of circle - 2 Marks, Function for Calculating area of rectangle - 2 Marks)

Ans:

```
#include<iostream.h>
#include<conio.h>
float area(float a)
{
return (3.14*a*a);
}
int area(int p,int q)
{
return(p*q);
}
void main()
{
clrscr();
cout<<"Area of circle:"<<area(6);
cout<<"Area of Rectangle:"<<area(5,6);
getch();
}
```

(e) Write any four features of object oriented programming.
(Any four features, each – 1 Mark)

Ans:

- Emphasis is on data rather than procedure.
- Programs are divided into what are known as objects
- Data structure designed such that they characterize the objects.
- functions that operate on the data of an object are tied together in the data structure.

- Data is hidden & cannot be accessed by external functions.
- Objects may communicate with each other's through functions.
- New data and functions can be easily added whenever necessary.
- Follows bottom-up approach in program designing.

(f) Explain pointer arithmetic with example.

(Description – 2 Marks, Example – 2 Marks; any other relevant program shall be considered)

Ans:

1. As a pointer holds the memory address of a variable some arithmetic operations can be performed with pointers. C++ supports four arithmetic operators that can be used with pointer such as:incrimination++
2. Pointers are variables. They are not integers, but they can be displayed as unsigned integers. The conversion specifier for a pointer is added and subtracted.

For example:

Ptr++: causes the pointer position to be incremented, but not by 1.

Ptr --: the pointer position to be decremented, but not by 1.

3. Following program segment shows the pointer arithmetic.

   The integer value would occupy bytes 2000 and 2001

```
int value, * ptr;
value=120;
ptr=&value;
ptr ++;
printf(%u\n", ptr);
```

Example:

```
#include<iostream.h>
#include<conio.h>
void main()
{
int value;
int*ptr;
ptr= &value;
cout<<"memory address before incrementation=";
cout<<ptr<<endl;
ptr++;
cout<<" memory address after incrementation=";
cout<<ptr<<endl;  }
memory address before incrementation=0X24c8ff4
memory address after incrementation=0X24c8ff5
```

4. Attempt any FOUR of the following: 16

(a) What is destructor? Give it's syntax. How many destructors can be defined in a single class?
(Definition - 2 Mark, Syntax - 1 Mark, Number of destructor in a class - 1 Mark; example is optional)

Ans: Destructor is used to destroy then objects that have been created by a constructor. Destructor is special member function whose name is same name as that of the class name but is preceded by tilde (~).

Syntax:

```
~Class_name();
```

e.g:

```
class integer
{
int m, n;
public:
~integer();
};
```

Only one destructor because we can't overload the destructor

(b) State different types of inheritance with diagram.

(Description – 1 Mark, Diagram - 1 Marks; any 4 types are expected)

Ans:

1. **Multiple Inheritance:**

A class can inherit the attributes of two or more classes as shown in fig .this is known as multiple inheritance.

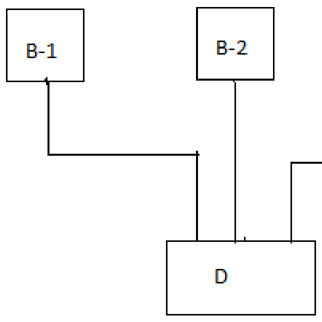
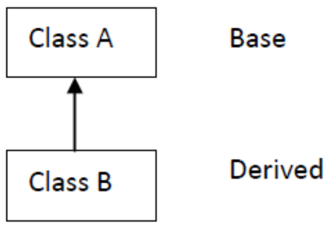


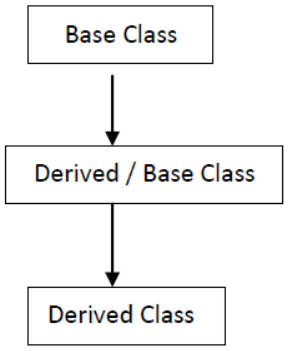
Fig. Multiple Inheritance

2. **Single inheritance**:

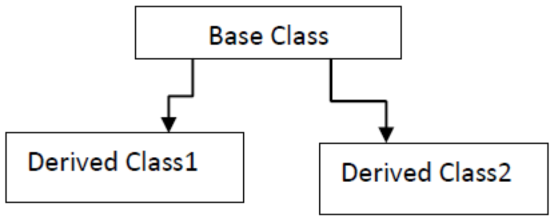
It includes single base class which can allow only one derived class to inherit its properties.



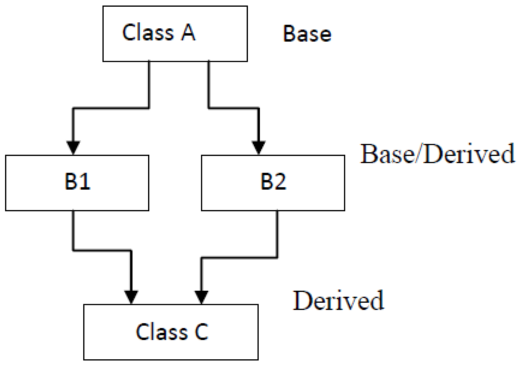
3. **Multi-level inheritance**: A single class can be derived from a single base class. We can derive a new class from as already derived class. It includes different levels for defining class. A child class can share properties of its base class (parent class) as well as grandparent class.



4. **Hierarchical inheritance**: In this inheritance, multiple classes can be derived from one single base class. All derived classes inherit properties of single base class.



5. **Hybrid Inheritance**: In this inheritance, it combines single inheritance, multiple inheritance, multi – level inheritance & hierarchical inheritance.



(c) Write a program to declare a class staff having data members as name and post. Accept and display data for five staff members. (Using array of object)

(Class definition - 2 Marks, Creation and working of array - 2 Marks)

Ans:

```
#include<iostream.h>

#include<conio.h>

class staff

{

private:

char name[10], post[10];

public:

void get()

{

cout<<"enter staff name and post"<<endl;

cin>>name>>endl>>post;

}

void put()

{

cout<<"Name of staff="<<name<<endl;

cout<<"Name of post="<<post;

}

};
```

```
void main()
{
staff s[5];
int i;
clrscr();
for(i=0;i<=4;i++)
{
s[i].get();
}
for(i=0;i<=4;i++)
{
s[i].put();
}
getch();
}
```

(d) Explain the concept of this pointer.
(Description - 3 Marks, Example - 1 Mark)

Ans:

1. C++ uses a unique keyword called *"**this**"* to represent an object that invokes a member function.
2. This is a pointer that points to the object for which this function was called.
3. This is a unique pointer is automatically passed to a member function when it is called.

4. The pointer this acts as an implicit argument to all the member function.
5. its an in-built pointer so programmer doesn't need to explicitly create this pointer.
6. one class can have exactly one this pointer.

Example:

```
class ABC
{
int a;
 …
voidgetdata()
{
}
 …};
```

The private variable 'a' can be used directly inside a member function, like

a=123;

We can also use the following statement to do the same job:

this->a=123;

this->getdata();

(e) State any four rules for operator overloading.

(Any four points; each - 1 Mark)

Ans: Rules for Overloading operators:

1. Only existing operators can be overloaded. Operatorscan not be created.
2. The overloaded operator must have at least one operand that is of user defined type.
3. We cannot change the basic meaning of an operator i.e. we cannot redefine the plus(+) operator to subtract one value from the other.
4. Overloaded operators follow the syntax rules of the original operators. They cannot be overridden. 5. There are some operators that cannot be overloaded. for e.g. sizeof, . , .* , : : , ?:
5. We cannot use friend functions to overload certain operators (=,( ),[ ],->).However member functions can be used to overload them.

6. Unary operators overloaded by means of a member function, take no explicit arguments and return no explicit values, but those overloaded by means of a friend function, take one reference argument.
7. Binary operators overloaded through a member function take one explicit argument and those which are overloaded through a friend function take two explicit arguments.

(f) Explain object as function argument.

(Description - 3 Mark, Example - 1 Mark, any relevant explanation shall be considered)

Ans: In object as function argument we send an object of a class to member function as an argument which allows programmer to share the data and functions associated with that object. This facilitates communication between two different classes. There are two types of having object as function arguments. As follows

Object as function argument: - where actual object is sent as an argument

Friend function: - where the reference of an object is sent as an argument to a function of another class.

Example:-

```
#include <iostream.h>
class rational
{
private:
intnum;
intdnum;
public:
void get ()
        {
cout<<"enter numerator";
cin>>num;
cout<<"enter denominator";
cin>>dnum;
        }
void print ()
        {
cout<<num<<"/"<<dnum<<endl;
        }
void multi(rational r1,rational r2) // receiving object as argument
        {
num=r1.num*r2.num;
dnum=r1.dnum*r2.dnum;
        }
};
void main ()
{
rational r1,r2,r3;
r1.get();
r2.get();
r3.multi(r1,r2); // sending object as an argument
r3.print();
}
```

5. Attempt any FOUR of the following: 16

(a) How many ways we can define member function in class? Give it's syntax.

(Out Side Class Definition - 2 Marks; Inside Class definition - 2 Marks)

Ans: Member functions can be defined in two ways:

- Outside class definition
- Inside class definition

- **Outside class definition :**
  - Member functions that are declared inside class have to be defined separately outside class. Their definitions are as simple as normal function definition. They should have function header & body

- Difference between member function & normal function is that member function incorporates membership 'identity label' in header
- General form of member function definition is as:

```
classclass_name
{
…
…
…
public:
return_typefunction_name(argument(s));
};
```

return-type class-name:: function-name(argument(s))

```
    {
        function body
    }
```

- Membership label class-name:: tell complier that function-name belongs to class class-name. This is scope of function restricted to class-name specified in header line. Scope resolution operator (::) is used.
- For example member functions getdata() &putdata() of class item can be coded as follows:

```
void item :: getdata(int a, float b)
{
        number =a;
        cost= b;
}
```

- **Inside Class Definition:**
  - This is method of defining member function is that member function declaration are replaced by actual function definition inside the class
  - For example item class can be defined as:

```
class item
{
    int number;                    //variable declaration
```

```
            float cost;                // private by default
            public:
            voidgetdata(inta, float b); //function declaration
            voidputdata ()             // function definition
               {
                       cout<<number<<"\n";
                       cout<<cost;
               }
        };
```

(b) Differentiate between structure and class.

(Any four points of comparison - 1 Mark each)

Ans:

Sr. No	**Structure**	**Class**
1	Structure is collection of variable of different data types	Class is collection of variables of different data types & functions
2	"struct" keyword is used while declaring structure	"class" keyword is used while declaring a class
3	Structure variables can be created	Class variables can be created which are called as objects of class
4	Access specifiers are not available	Access specifiers like private, public, protected are available
5	Data hiding is not achieved	Data hiding is achieved with private keyword

6	Syntax:
	structStruct_name
	{
	variable 1;
	variable 2;
	…………..
	variable n;
	}struct_var;
		Syntax:
		class class_name
		{
		private:
		member variable declaration;
		member function declaration;
		public:
		member variable declaration;
		member function declaration;
		};

(c) Why do we need virtual functions?

(Need - 4 Marks, Example optional)

Ans:

- Polymorphism refers to property by which objects belonging to different classes are classes are able to respond to same message, but in different forms.
- Therefore an essential requirement of polymorphism is ability to refer to object without any regard of their classes. This requires use of single pointer variable to refer to objects of different classes.
- Here we use pointer to base class to refer to all derived objects. Base pointer even when it is made to contain address of derived class, always executes function in base class.
- Compiler simply ignores content of pointer & chooses member function that matches type of pointer. In this case polymorphism is achieved by using virtual functions.
- When we use same function name in both base & derived classes, function in base class is declared as virtual using keyword virtual preceding its normal declaration.
- When function is made virtual C++ determines which function to runtime based on type of object pointed to by base pointer rather than type of pointer.
- Thus by making base pointer to point to different objects different version of virtual function can be executed.
- Runtime polymorphism is achieved only when virtual function accessed through pointer to base class.
- **Example:**

```
//Virtual function
#include<iostream.h>
#include<conio.h>
class base
{
    public:
    void display()
    {   cout<<"\n Display Base";     }
    virtual void show()
    {    cout<<"\n Show Base";          }
};
class derived : public base
{
    public:
    void display()
    {    cout<<"\n Display Derived";  }
    void show()
    {    cout<<"\n Show Derived";     }
};
void main()
{
    base B;
    derived D;
    base *bptr;
    clrscr();
    cout<<"\n bptr points to base \n";
    bptr=&B;                    //pointer to base object
    bptr->display();            //calls base version
    bptr->show();               //calls base version
    cout<<"\n bptr points to derived \n";
    bptr=&D;
    bptr->display();            //calls base version
    bptr->show();               //calls derived version
}
```

(d) Write a program to search a number from an array using pointer to array.

(Correct Logic - 2 Marks, syntax – 2 Marks)

Ans: CODE:

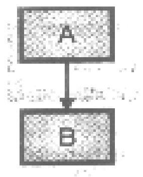
```
#include<iostream.h>
```

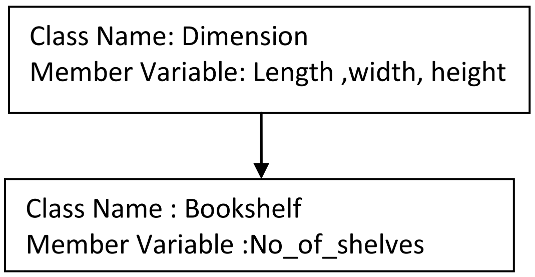
```
#include<conio.h>
void main()
{
        int a[5], i,*a1, no, flag=0;
        clrscr();
        a1=&a[0];
        cout<<"\n Enter array elements: \n";
        for(i=0; i<5; i++)
        {
            cout<<"Enter "<<i<<" elements: \n";
            cin>>*a1;
            a1++;
        }
        cout<<"Enter element to be searched:\n";
        cin>>no;
        a1=&a[0];
        for(i=0; i<5; i++)
        {
                if(*a1==no)
                {
                    cout<<"Number is present at "<<i+1<<" Position.\n";
                    flag++;
                    a1++;
                    }
                else
                {
                    a1++;
                }
        }
        if(flag == 0)
        {       cout<<" Number is not present.\n";    }
getch();
}
```

(e) Explain single inheritance with program.

(Description with diagram - 2 Marks; Program - 2 Marks; Any Relevant Program Shall Be Considered)

Ans: Derived class with only one base class is called single inheritance.




Class Name: Dimension
Member Variable: Length ,width, height
Class Name : Bookshelf
Member Variable :No_of_shelves

Program :-

```
#include<iostream.h>
#include<conio.h>
class dimension
{
    intlength,width, height;
    public:
    voidgetdata()
    {
        cout<<"Enter length, width ,Heigth\n";
        cin>>length>>width>>height;
    }
    void display()
    {
        cout<<"Length:"<<length<<endl<<"Width:"<<width<<endl<<"Height:"<<height;
    }
};
classbookshelf: public dimension
{
    intno_of_shelves;
    public:
    void accept()
    {
        cout<<”Enter No_Of_Shelves:”;
        cin>>no_of_shelves;
    }
    void show()
    {
        cout<<” No_Of_Shelves:”<<no_of_shelves;
    }
};
void main()
{
    bookshelf b;
    b.getdata();
    b.display();
    b.accept();
    b.show();
getch();
}
```

(f) Explain constructor with default argument.
(Description - 3 Marks, Example - 1 Mark)

Ans:

- A constructor is special member function whose task is to initialize objects of it’s class. It is special because its name is same as that of class name.

- Constructor is invoked whenever an object of its associated class is created. It is called constructor because it constructs values of data members of the class.

Constructors With Default Argument:

- It is possible to define constructors with default argument. For example constructor complex() can be declared as follows:
  complex (float real , float imag =0);

  Default value of argument imag is zero. Then statement

  Complex C (5.0);

  Assigns value 5.0 to real variable & 0.0 to imag (by default)

  Statement
  Complex C (2.0,3.0);

  Assigns 2.0 to real & 3.0 to imag

- Actual parameter when specified overrides default value. Missing argument must be trailing arguments
- “Default constructor” A :: A() is totally different than “Constructor with default argument” A :: A (int = 0)
- Default argument constructor can be called with either one or no argument. When called with no argument it becomes default constructor
- When both these forms are used in class it causes ambiguity for statement such as
  A a;
  Ambiguity is whether to call A :: A() or A :: A(int = 0)

6. Attempt any TWO of the following: 16

(a) Identify the type of inheritance shown in following fig-1. Implement it by using suitable member function.

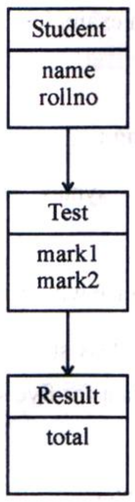


Fig.-1

(Type Identification - 2 Marks, Implementation - 6 Marks)

Ans: Type of inheritance: **Multilevel Inheritance**

CODE:

```
#include<iostream.h>
#include<conio.h>
class student
{
    protected:
    introll_no;
    char name[15];
    public:
    void get()
    {
        cout<<"\n Enter Roll no & name";
        cin>>roll_no>>name;
    }
void show()
    {
        cout<<"\n Roll_no:"<<roll_no;
        cout<<"\n Name:"<<name;
    }
    };
classtest:public student
{
    protected:
    int mark1,mark2;
    public:
void get1()
{
cout<<"\n Enter marks of 2 subjects";
cin>>mark1>>mark2;
}
void show1()
{
cout<<"\n Marks Are: \n1)"<<mark1<<"\n2)"<<mark2;
}
};
classresult:public test
{
protected:
int total;
public:
voidcal()
{
total=mark1+mark2;
cout<<"\n Total is:"<<total;
```

```
}
};
void main()
{
result R;
R.get();
R.show();
R.get1();
R.show1();
R.cal();
getch();
}
```

(b) Write a program to declare a class birthday having data members day, month, year. Accept this information for five objects using pointer to the array of objects.

(Implementation of Class with Appropriate Functions - 4 Marks, Function Call - 4 Marks)

Ans:

```
#include<iostream.h>
#include<conio.h>
class birthday
{
        intday,month, year;
        public:
        voidgetdata()
        {
                cout<<" Enter birthdate(day month year):\n";
                cin>>day>>month>>year;
        }
        voidputdata()
        {
                cout<<" "<<day<<"/ "<<month<<"/ "<<year;
        }
};
void main()
{
        int i;
        birthday B[5], *ptr;
        clrscr();
        ptr= &B[0];
        for(i=0; i<5;i++)
        {
                cout<<"Enter birthdate for object;"<<i+1;
                ptr->getdata();
                ptr++;
        }
        ptr= &B[0];
        for(i=0; i<5;i++)
        {
                cout<<"\nBirthdate for object: "<<i+1;
                ptr->putdata();
                ptr++;
        }
getch();
}
```

(c) Explain overloaded constructor in a class with suitable example.

(Description of Overloaded Constructor - 4 Marks, Example - 4 Marks, any relevant example shall be considered)

Ans:

- There are no argument constructor, one argument constructor & even parameterized constructor

- C++ permits to use all these constructors in same class.
- Example:

```
class integer
{
        intm,n;
        public:
        integer()
        {
        m=0;
        n=0;
        }                       //constructor 1
        integer(int a, int b)
        {
 m=a;
 n=b;
}                               //constructor 2
        integer(integer &i)
        {
 m=i.m;
n=i.n;
}                       //constructor 3
};
```

- This declares three constructors for an integer object. First constructor receives no argument, second receives two integer argument & third receives one integer object as an argument
- Statement integer I1; invokes first constructor & set both m & n of I1 to zero
- Statement integer I2(20,40); invokes second constructor & set both m & n of I2 to 20 & 40 respectively
- Statement integer I3(I2); invokes third constructor which copies values of I2 into I3

- When more than one constructor function is defined in class then we say that constructor is overloaded.

- **Program:**

```
#include<iostream.h>
#include<conio.h>
classoop
 {
inta,b;
public:
oop()
    {
        a=100;
        b=50;
    }
oop(int x,int y)
    {
        a=x;
        b=y;
    }
voidputdata()
    {
        cout<<"\nValue 1"<<a;
        cout<<"\nValue 2"<<b;
    }
 };
void main()
 {
intp,q;
oop o1;
oop o2(50,100);
clrscr();
o1.putdata();
o2.putdata();
getch();
 }
```